# Data structures

Big Data
Kristiania University College

## Objectives

1. Understand the need to store information in data structures.

2. Understand the need to use the type of data structure most appropriate according to data processing to be performed in the script.

3. Know how to use the different types of existing data structure in Python.

# Table of Contents

# Data structures

## Introduction

Programming is about information representation.

- Simple data are easy to represent: Numbers, characters, strings, etc.

Reality uses to be more complicated.

- A class represent an object.
- How can we store several objects?
- How can we represent complex data?

We need powerful mechanisms to store information: Data structures.

# Java Collections

## Array

### Vector (1-D array)

|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| | $a_0$ | $a_1$ | $a_2$ | $a_3$ |

### Matrix (2-D array)

|  | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | $a_{0,0}$ | $a_{0,1}$ | $a_{0,2}$ | $a_{0,3}$ |
| 1 | $a_{1,0}$ | $a_{1,1}$ | $a_{1,2}$ | $a_{1,3}$ |
| 2 | $a_{2,0}$ | $a_{2,1}$ | $a_{2,2}$ | $a_{2,3}$ |

Advantajes:

- Very fast

Disadvantajes:

- Fixed size
- Nor supported in Python by default
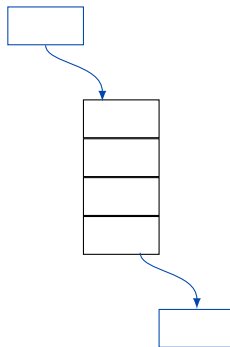
Kristiania
University
College

# Data structures

## Data structures (I): Stack and queue
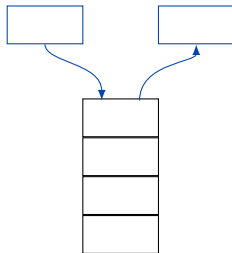


Stack (LIFO)          Queue (FIFO)

Operations:

- push(value) and pop(value)

Implemented as lists in Python

Data structures
○○○○●○○○
Data structures in Python
○○○○○○○○○
Other data structures in Python
○○○○○○○○○○○○○
Summary
○

# Data structures

## Lists and hash tables

Lists



Operations:

- `insert(pos, value)`
- `get(pos)`

Hash table
(associative array, dictionary)

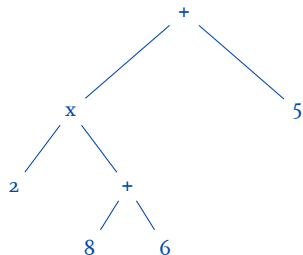| Key 1 | Value 1 |
| Key 2 | Value 2 |
| Key 3 | Value 3 |
| Key 4 | Value 4 |

Operations:

- `put(key, value)`
- `get(key)`

Data structures

Big Data

Kristiania
University
College

7 / 32

# Data structures
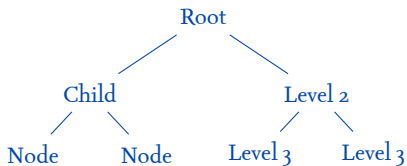
## Trees (I)

Trees



Operations:

- `insert()` and `remove()`
- `search()`

$$2 * (8 + 6) + 5$$

Kristiania
University
College

Data structures

Big Data

8/32

# Data structures

## Trees (II)
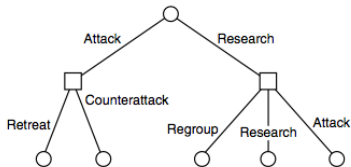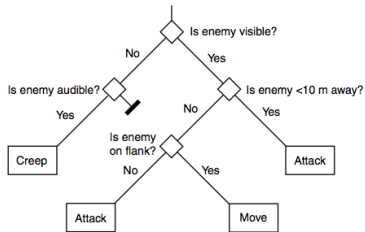


Source: Ian Millington, John Funge. "Artificial Intelligence for Games". Ed. Morgan-Kaufmann. 2009.
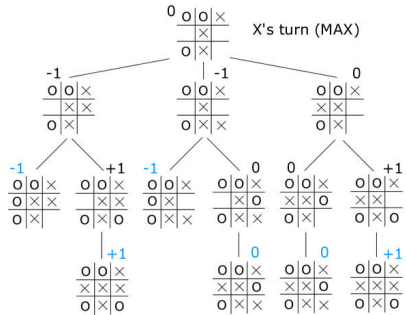
# Data structures

## Graphs

### Graphs
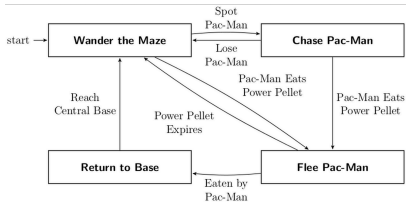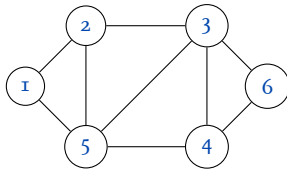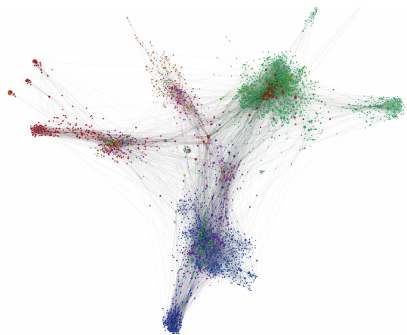




(Source)

(Video Path-Planning)

Kristiania
University
College

Data structures

Big Data

10/32

# Data structures in Python

## Overview

High-level, language-defined data structures:

- Lists.

- Tuples and sequences.

- Sets.

- Dictionaries (associative arrays).

# Data structures in Python

Lists (I)

Methods:

- `list.append(x)`
- `list.insert(i, x)`
- `list.remove(x)`
- `list.pop()`
- `list.index(x)`
- `list.count(x)`
- `list.sort()`
- `list.reverse()`

### List initialization

```
list = [item1, ..., itemN]
```

Lists are objects

Kristiania
University
College

# Data structures in Python

## Lists (II)

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Kristiania
University
College

Data structures

Big Data

13/ 32

# Data structures in Python

Lists (III)

## Slice notation in lists

```
t = [0, 1, 2, 3]
print(t)
print(len(t))
print(t[1])
print(t[1:3])
print(t[2:])
print(t[-1])
print(t[:-1])
print(t[:-3])
print(t[::-1])
```

# Data structures in Python

## Lists (IV)

Sometimes it is useful to `split` a string to build a list (split) and, conversely, `join` the elements of a list to build a string

**join-split.py**

```python
cadena_ejemplo="Cadena para prueba de join y split"

print (cadena_ejemplo.split())
print ("otra-prueba".split("-"))

con_lista=["Cadena2", "de", "prueba", "de", "join"]

#print (con_lista.join()) # ERROR!
print("".join(con_lista))
print(",".join(con_lista))
```

# Data structures in Python

## Lists as stacks

Just use two methods: `append()` and `pop()`

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Kristiania
University
College

# Data structures in Python

## Lists as queues

Queues with lists is not very efficient

- Use instead the `deque` module from the `collections` library.

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")
>>> queue.append("Graham")
>>> queue.popleft()
'Eric'
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
```

New Python feature: Modules

Kristiania
University
College

Data structures

Big Data

17/ 32

# Data structures in Python

`del` is used to delete items and variables

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  NameError: name 'a' is not defined
```

New Python feature: Error traces

# Other data structures in Python
## Tuples (I)

**Tuple:** A sequence of *ordered* items, very similar to lists.

- However they are not the same.
- Lists are *mutable*, tuples are *inmutable*.
- Tuples use to contain, **usually**, heterogeneus items.
- Lists use to contain, **usually**, homogeneus items, used to iterate.
- Lists and tuples are ordered

### Creation

```
tup1 = 1, 2, 3
tup2 = ("Hi", 1.1, 2)
tup3 = (0, (1, 3), 2)
```

### Manipulation

```
>>> tup1[0]
1
>>> tup1
(1, 2, 3)
>>> tup1[1:]
(2, 3)
```

Kristiania
University
College

# Other data structures in Python

## Tuples (II)

**Modification**

```
>>> tuple1 = ('a', 'z', 'c')
>>> tuple1[0] = 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tuple1.append('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError:'tuple' object has no attribute 'append'
>>> tuple1.index('z')
1
>>> () == True
False
```

# Other data structures in Python

## Sets (I)

**Set:** A collection of items, unordered with no duplicates.

- Membership testing.
- Eliminating duplicate entries.
- Math operations: `union()`, `intersection()` and `difference()`.

### Creation (I)

```python
set1 = {"red", "blue"}
>>> type(set1)
<class 'set'>
>>>  set1 = set()
>>> set1
set1()
>>> what_is = {}
>>> type(what_is)
<class 'dict'>
```

### Creation (II)

```python
list_mix = ['a', True, 33]
>>> set_mix =  set(list_mix)
>>> set_mix
{'a', True, 33}
>>>  len(set_mix)
3
>>> 33 in set1
True
```

Kristiania
University
College

Data structures

Big Data

21/32

# Other data structures in Python

Sets (II). Modification

```
set_mix1 = {'a', 'b'}
>>> set_mix1.add('c')
{'a', 'b', 'c'}
>>> set_mix1.add('a')
>>> set_mix1
{'a', 'b', 'c'}
>>> set_mix1.update({'b', 'c', 'd'}, {'b', 'e', 'a'})
>>> set_mix1
{'a', 'b', 'c', 'd', 'e'}
>>> set_mix1.update(['b', 'c', True])
>>> set_mix1
{'a', 'b', 'c', 'd', 'e', True}
>>> set_mix1.discard(False)
>>> set_mix1
{'a', 'b', 'c', 'd', 'e', True}
```

Data structures

Big Data

Kristiania
University
College

22/32

# Other data structures in Python

```
>>> set_mix1.remove(False)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: False
>>> set_mix1.remove(True)
>>> set_mix1
{'a', 'b', 'c', 'd', 'e'}
>>> set_mix1.pop()
'c'
>>> set_mix1
{'a', 'b', 'd', 'e'}
>>> set_mix1.clear()
>>> set_mix1
set()
>>> set_mix1 = {2, 5}
>>> set_mix2 = {1, 2, 3}
>>> set_mix1.union(set_mix2)
{1, 2, 5, 3}
```

**Sequence**: All types that behaves like sequences: Strings, lists and tuples.

# Other data structures in Python

## Dictionaries (I)

**Dictionary**: A collection of pairs `<key, value>`

- Also named as *associative array*, very similar to hash maps.
- Lists are indexed with a number, dictionaries use keys.
- Key: Numbers, strings, tuples and any inmutable type.

### Creation

```
>>> tel = {'jack' : 4098, 'sape'
    : 4139 }
>>> tel['guido'] = 4127
>>> tel
{'guido': 4127, 'jack': 4098, '
    sape': 4139}
```

### Manipulation

```
>>> del tel['sape']
>>> tel
{'guido': 4127, 'jack': 4098}
>>> list(tel.keys())
['guido', 'jack']
>>> 'guido' in tel
True
```

Data structures

Big Data

Kristiania
University
College

24/ 32

# Other data structures in Python

## Dictionaries (II)

Dictionaries can be iterated by key or by value

- Loop syntax is slightly different
- `item()` method

### Dictionary iteration

```
knights = {'gallard' : 'the pure', 'robin' : 'the brave'}
for k, v in knights.items():
  print(k, v)
```

Kristiania
University
College

Data structures
○○○○○○○○

Data structures in Python
○○○○○○○○

Other data structures in Python
○○○○○○○●○○○○○○

Summary
○

# Other data structures in Python
## Looping techniques (I)

A bunch of useful functions for looping

| | |
|---:|---|
| `enumerate()` | Retrieve position index and value. |
| `zip()` | Pair two or more sequences. |
| `sorted()` | Iterate in order. |
| `reversed()` | Iterate in reverse order. |

# Other data structures in Python

## Looping techniques (II)

### enumerate()

```
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)
```

### zip()

```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']

for q, a in zip(questions, answers):
    print('What is your {0}? It is {1}.'.format(q, a))
```

Kristiania
University
College

# Other data structures in Python

## Looping techniques (III)

### sorted()

```python
basket = ['apple', 'orange', 'apple', 'pear']
for f in sorted(set(basket)):
    print(f)
```

### reversed()

```python
for i in reversed(range(1, 10, 2)):
    print(i)
```

# Other data structures in Python

## More on conditions (I)

Comparison operators

| | |
|---|---|
| == | Equal to |
| != | Not equal to |
| <> | Similar to != |
| | (deprecated in 3.x) |
| > | Greater than |
| < | Less than |
| >= | Less or eq. to |
| <= | Less or eq. to |

Conditional operators

| | |
|---|---|
| and | AND |
| or | OR |
| not | Negation |

- Widely used in loops and conditions
- Result: `true` or `false`
  - Python supports boolean variables
  - The result is a boolean
- Truth tables represent the conditional operators

Truth tables

| A | TTFF |
|---|---|
| B | TFTF |
| A and B | TFFF |

| A | TTFF |
|---|---|
| B | TFTF |
| A or B | TTTF |

# Other data structures in Python

## More on conditions (II)

### Example

```python
value1 = int(input("Give me a number:"))
value2 = int(input("Give me another number:"))

if value1 == value2:
  print("value1 == value2")
else:
  print("value1 != value2")

if value1 > value2:
  print("value1 > value2")
elif value1 < value2:
  print("value1 < value2")
```

Kristiania
University
College

Data structures

Big Data

30/32

# Other data structures in Python

## More on conditions (III)

| Identity operators | |
| --- | --- |
| is | Same objects |
| is not | Not same objects |

| Membership operators | |
| --- | --- |
| in | Contained |
| not in | Not contained |

- Identity operators compare `objects`
  - We will study objects later, do not worry right now
- Membership valid on sequences
  - Remember: A sequence is a string, tuple or list

### Example

```
value = int(input("Give me a number between 1 and 5:"))

while value not in range(1, 6):
    value = int(input("Give me a number between 1 and 5:"))
```

Kristiania
University
College

Data structures

Big Data

31/32

# Summary

|  | MUTABLE | ORDERED | INITIALIZATION |
|---|---|---|---|
| List | Yes | Yes | `li = [1, 2, 3]` |
| Tuple | No | Yes | `tu = (1, 2, 3)` |
|  |  |  | `tu = 1, 2, 3` |
| Set | No | No | `se = {1, 2, 3}` |
| Dictionary | Yes | No | `dic = {'abc' : 1, 'bca' : 2}` |

Kristiania
University
College